DASH in Twitch: Adaptive Bitrate Streaming in Live Game Streaming Platforms

Karine Pires, Gwendal Simon

DASH in Twitch: Adaptive Bitrate Streaming in Live Game Streaming Platforms

Karine Pires
Telecom Bretagne, France
karine.pires@telecom-bretagne.eu

Gwendal Simon
Telecom Bretagne, France
gwendal.simon@telecom-bretagne.eu

ABSTRACT

Live game streaming platforms such as Twitch allow gamers to broadcast their gameplay over the Internet. The popularity of these platforms boosts the market of eSport but poses new delivery problems. In this paper, we focus on the implementation of adaptive bitrate streaming in massive live game streaming platforms. Based on three months of real data traces from Twitch, we motivate the need for an adoption of adaptive bitrate streaming in this platform to reduce the delivery bandwidth cost and to increase QoE of viewers. We show however that a naive implementation requires the reservation of a large amount of computing resources for transcoding purposes. To address the trade-off between benefits and costs, we formulate a management problem and we design two strategies for deciding which online channels should be delivered by adaptive bitrate streaming. Our evaluations based on real traces show that these strategies can reduce the overall infrastructure cost by 40% in comparison to an implementation without adaptive streaming.

Keywords

Live streaming; user-generated content; video encoding

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network monitoring*

1. INTRODUCTION

The market of video game competition (also known as *eSport*) has been boosted by the rise of online live streaming platforms [5]. Every month in 2013, around one million gamers have broadcasted themselves playing games live, and more than 40 millions of people have watched these gameplay video channels [2]. This popularity has made the leading live game streaming platform, namely Twitch[1], become the fourth largest source of US peak Internet traffic in February 2014 [6]. The rapidly rising bandwidth need of live game streaming platforms requires infrastructure updates and calls for the development of dedicated delivery solutions.

In the recent years, video providers have deployed Adaptive Bitrate (ABR) streaming to cope with the heterogeneity of devices and network connections. This technique applied to live streams is also adopted by the online services of TV companies [9]. However, to the best of our knowledge, the implementation of ABR in live streaming services is limited to a small number of video channels in dedicated servers. In the case of massive live streaming platforms such as Twitch, both the large number of concurrent channels and the use of commodity servers in data-centers pose new problems.

Implementing ABR in massive live streaming platforms yields some benefits and costs. On the negative side, for each video channel, the raw live video stream should be transcoded into multiple live streams at different resolutions and bitrates. The consumption of computing resources needed for the transcoding induces significant costs, especially with regards to the large number of concurrent video channels. On the other hand, the benefits include the improvement of the Quality of Experience (QoE) for the end-users and a reduction of the delivery bandwidth costs.

In this paper, we study the trade-off between benefits and costs for the implementation of ABR in live game streaming platforms. First, we analyze three months of real traces from Twitch. We highlight some key characteristics of this platform in Section 2. Second, we identify that one key problem is to decide which video channels should be broadcasted as usual (i.e. by directly forwarding to the viewers the raw video received from the broadcaster) and which channels should be delivered with ABR streaming (see Section 3.1). Third, we present in Section 3 two strategies, according to whether the decision of delivering a given channel by ABR can be taken while this channel is online (*on-the-fly strategies*) or only when the channel starts broadcasting (*at-the-startup strategies*). Finally, we compare in Section 4 both strategies in a realistic scenario based on our real traces from Twitch and a dataset for the population [3].

By extension, our work applies to any live streaming platform without regard to the type of the broadcasted videos. This paper is also part of a wider study related to video transcoding in the cloud, more specifically in data-centers where resources are shared, commoditized servers. We discuss these points in Section 5.

[1] `http://twitch.tv`

VideoNext'14, December 2, 2014, Sydney, Australia.

http://dx.doi.org/10.1145/2676652.2676657.

2. INTRODUCING TWITCH

In Twitch we distinguish *broadcasters* and *viewers*. The broadcasters are registered gamers, who are in charge of one *channel*. We will interchangeably use the terms channel and broadcaster hereafter. A channel can be either *online* at a given time, which means that the broadcaster is broadcasting a gameplay live, or *offline* when the user is disconnected. A channel can alternatively switch from offline to online and vice versa. When a channel is online, we say that it corresponds to a *session*. The number of *viewers* watching a session can change over the time of the session. We illustrate in Figure 1 the evolution of the popularity of a given channel over time, this channel containing two sessions.

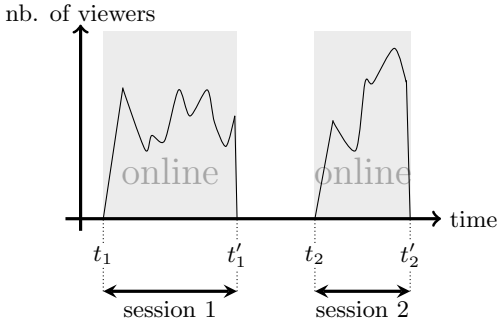


Figure 1: A life in a channel

2.1 Twitch Dataset

Twitch provides an Application Programming Interface (API) that allows anybody to fetch information about the state of Twitch. We used a set of synchronized computers to obtain a global state every five minutes (in compliance to API restrictions) between January 6 and April 6, 2014. We fetched information about the total number of viewers, the total number of concurrent online channels, the number of viewers per session, and some channels metadata. We then filtered the broadcasters having abnormal behavior (*e.g.* no viewer over the three months or only one five-minute session). The dataset, containing more than five millions sessions, is available on a public website.[2]

2.2 Characterizing Twitch

We give in the following a short analysis of the main characteristics of Twitch. We represent the minimum and maximum values per day that we observed from the 288 daily measures.

Delivery Needs. We evaluate the overall bandwidth needed to deliver video channels to the viewers (we do not take into account the bandwidth required from the broadcasters to Twitch data-center). Twitch is a regular Over-The-Top (OTT) service with unicast transmission to viewers, so we sum up the bitrates of each session multiplied by the number of viewers for this session. We see in Figure 2a that the daily bandwidth peak is often more than 1.5Tbps with a peak at more than 2Tbps. Moreover, the delivered traffic is sustained with minimum daily bandwidth always above 400Mbps.

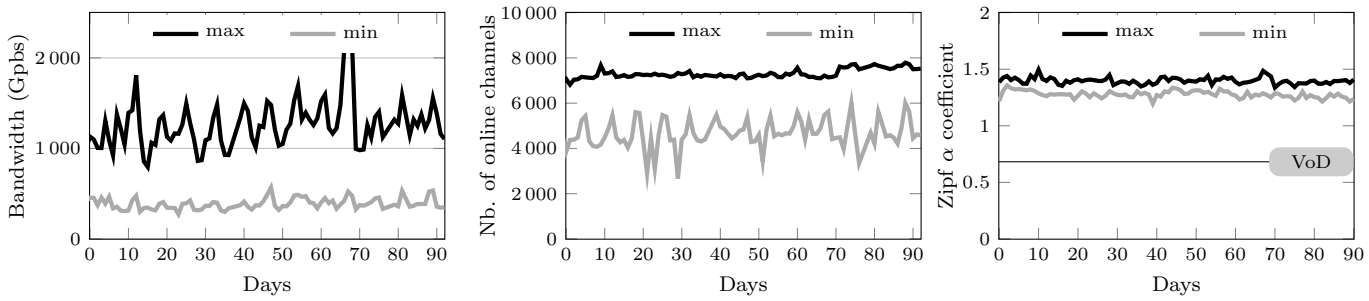
[2] `http://dash.ipv6.enstb.fr/dataset/videonext-2014/`

Computing Needs. Another infrastructure cost is the data-center, which, among others, processes the incoming raw video from broadcasters and prepares the streams to be delivered. We present in Figure 2b the average number of concurrent online channels, which is a metric for estimating the data-center dimensions. Between 4,000 and 8,000 concurrent sessions always require data-center processing. Such sustained incoming traffic requires a computing infrastructure that, to our knowledge, is unique in the area of live streaming.

Channel Popularity. The distribution of popularity in User Generated Content (UGC) platforms typically follows the Zipf law [7]. We first need to check whether the popularity of Twitch broadcasters follows a Zipf law as well. Considering the Equation 1 we produce an approximation of the Zipf parameters using a fitting curve process on the R software. We validate the results of the approximation by calculating the Normalized Root-Mean-Square Deviation (NRMSD) between the real data and the fitting curve. The mean NRMSD value is 0.0095 with confidence intervals lesser than 1%. In other words, broadcaster popularity in Twitch follows a Zipf law. We then analyze the value of the α parameter, which says how much heterogeneous is the popularity of broadcasters. The larger α is, the more heterogeneous is the platform. Figure 2c shows the results obtained for the Zipf α coefficient. The horizontal line indicates the value found on YouTube [7]. What is relatively surprising here is the high value of α for Twitch. Although α is often lower than 1 in other UGC platforms, it is always larger than 1.3 over the three months, and even sometimes above 1.5. Such a large α coefficient characterizes both a sharp difference between the most popular channels and the others, and a long tail of unpopular channels.

$$F_i(x) = A_i x^{-\alpha} \tag{1}$$

Raw Videos. The raw live stream is the video encoded at the broadcaster side and transmitted to the data-center of Twitch. This video can be encoded in various resolutions and bitrates. Our dataset confirms the intuitive idea that quality and popularity are connected. It is likely to be a circular dependency where the better is the quality of the video, the more attractive to viewers and the more popular it becomes. And by being more popular and successful, broadcasters would be more willing to increase the quality of their videos by, for example, investing on better equipment. In the Figure 3, we show the ratio of sessions with raw video at a given resolution, as well as the ratio of viewers on a channel with a video at this resolution. The sessions with videos at a resolution lesser than 720p represent 40% of the total amount of sessions but they attract only 8% of the total viewers. Figure 4 shows the Cumulative Distribution Function (CDF) of the bitrates of sessions for the three most popular resolutions. The bitrates of 720p and 1080p channels are significantly higher than for 480p channels. To emphasize the gap, we draw a thin vertical line at 2 Mbps. Only half of the video sessions at both 720p and 1080p have a bitrate lower than 2Mbps although such a bitrate is larger than 90% of the bitrates of 480p channels.



(a) Bandwidth usage for live video delivery (b) Number of concurrent online channels (c) Zipf α coefficient

Figure 2: Three months in Twitch

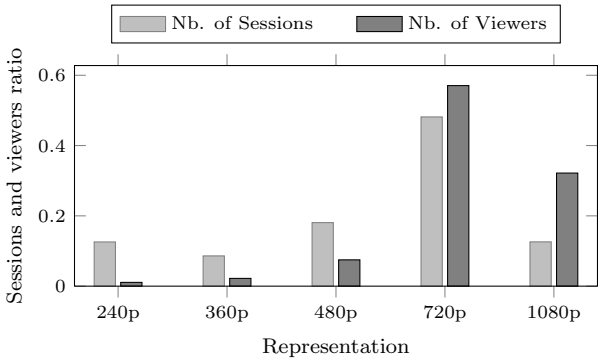


Figure 3: Number of sessions and viewers by video representation

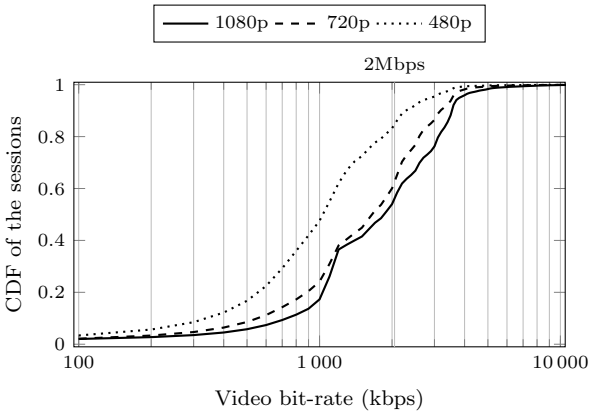


Figure 4: CDF of the session bitrates

3. WHICH CHANNELS TO TRANSCODE

We extract three main ideas from the measurements of Section 2.2. First, ABR streaming is required to cope with high delivery cost and with the inaccessibility of the most popular channels to a fraction of the population. Second, applying ABR to all channels requires a significant amount of computing resources because the number of concurrent sessions is high. Third, all channels should not be treated equally since only a few ones are popular. In the following, we introduce the problem of deciding the subset of channels to which ABR should apply. We then present two strategies.

3.1 Trade-off and Problem Definition

We distinguish two types of process when the raw live video of an online channel is received by Twitch:

- The *traditional process* consists of preparing the raw live stream (e.g. for sanity check and better webpage integration), and then delivering it directly to the viewers requesting it.
- The *transcoding process* consists of transcoding the raw live stream into multiple live video streams and of using ABR streaming to deliver the session to the viewers, typically with a standardized technology [1].

We envision that only a fraction of online channels should use ABR streaming: the popular channels with high bitrate and resolution. For a given channel, ABR streaming generates two main benefits. First, it improves the QoE for all the viewers having a downloading rate inferior to the raw video bitrate. We call them the *degraded viewers*. Without ABR, the degraded viewers experience video buffering at a frequency that depends on the difference between their downloading rate and video bitrate. Such a bad QoE causes churn and degrades the reputation of the platform. Second, ABR streaming reduces the overall needed bandwidth for serving a population. Without ABR, Twitch should deliver the raw live video to all viewers. On the contrary with ABR, the degraded viewers are served with a video stream at a lower bitrate than the raw live stream.

We distinguish two cases for the degraded viewers: (*i*) the viewer has his video session delayed in time but continues to watch, therefore the overall amount of data transferred is the same as any other viewer; (*ii*) the viewer experiences frequent video buffering and is likely to quit, meaning no more bandwidth will be used to deliver the video for that viewer. Since our data has the current number of viewers watching a channel, when a viewer quits a channel this viewers counter is decreased. Thus viewers that quit are discarded, which leads to viewers considered in this work are either satisfied viewers or degraded viewers in the first case.

The problem is to decide which process should apply for every online channel at a given time with respect to the trade-off between extra-cost induced from the transcoding process and the gains. To make it harder, switching from one process to another *while the channel is online* is not trivial. A possible implementation is to use ABR streaming technologies for both decisions. The *representation set* of a

channel in the transcoded process contains the multiple live video streams while it contains only the raw live stream for channels in the traditional process. Thus, switching from one process to another *on-the-fly* requires a revision of the *manifest* with the new set of video representations and the notification of all users about the manifest revision. Both actions are not always possible with respect to the ABR streaming technology.

Due to this uncertainty, we distinguish:

- *On-the-fly strategies*, where the decision of whether an online channel should be transcoded or not can be taken at anytime during a session. In Figure 1, an on-the-fly strategy can decide to transcode or not the channel at anytime between t_1 and t'_1 and between t_2 and t'_2.
- *At-startup strategies*, where the decision of whether an online channel should be transcoded or not can be taken only when the session starts. In Figure 1, the decision at time t_1 applies for the whole session 1 and the decision at time t_2 for the whole session 2.

In the following, we present a simple implementation for each strategy type. Our goal is to keep the strategy as simple as possible for a fast implementation.

3.2 On-the-Fly Strategy

An on-the-fly strategy is expected to perform well since it is reactive to any event, including unexpected events like flash crowd on a video channel or abrupt disinterest for another. That is, it is possible to build a strategy that is close to the optimal (with respect to the ability of the platform provider to quantify the QoE of degraded viewers). In this paper, our goal is not to define the best strategy but rather to highlight the performances of on-the-fly strategies in general. We thus design a simple strategy as follows.

First, we filter the broadcasters so that only broadcasters with a raw live video at resolutions 480p, 720p, and 1080p are considered as candidates for ABR streaming. Then, we define a value j, which is the threshold on the number of viewers. Every five minutes, all candidate broadcasters with more than j viewers are selected to be transcoded. We call this strategy *threshold-j*.

3.3 At-Startup Strategy

An at-startup strategy requires foreseeing the popularity of a channel in the near future. Predicting video popularity has become an important research topic with connection to massive data treatment, artificial intelligence, and social network observations. However, with the same motivations as for the on-the-fly strategy, we rather stay simple in this paper and left for future works the design of sophisticated efficient strategies.

To predict the popularity of channels, we only base on the history. The strategy comes from the observation that a channel that has been popular in the past will be popular in the future. To estimate how popular was a past session with respect to the context at that time, we focus on a simple measure: the top-k channels, *i.e.* the k most popular channels. Every five minutes, we collected the k most popular channels. Overall, for each month, we gathered more than 8,500 different sets of top-k channels. We analyzed these sets and observed that the number of *distinct* broadcasters is low. Typically for top-10 sets, around 500 broadcasters occupy the over 85,000 "spots" that are available every month.

We derive from this observation an at-startup strategy, which we call *top-k*. On a periodic basis, we get the k most popular broadcasters and we insert them to a list of candidate broadcasters. When a new session starts, the decision to apply the transcoding process to the broadcaster is taken if the broadcaster is in the list of candidates and if the raw live video has a resolution 480p, 720p, or 1080p. Note that we do not implement any mechanism for removing broadcasters from the candidate list after a while, so this list continuously inflates with time but such a mechanism is trivial to implement.

4. EVALUATION

We now evaluate the performances of both threshold-j and top-k strategies and also show the feasibility of the implementation of ABR for Twitch.

4.1 Settings

We use two real datasets for the setting of our evaluation. The first dataset is our Twitch dataset as described in Section 2. One time unit is five minutes, as the time needed to refresh the API at Twitch. We use the information of sessions (video bitrate, video resolution) and broadcasters (number of viewers). Simulations were done with three months of data and the figures present results of the last month.

Population Download Rate Settings. The second dataset comes from [3]. Since the Twitch API does not provide any information about the viewers (neither their geographic positions, nor the devices and the network connections), we need real data to set the download rates for the population of viewers. The dataset presented in [3] gives multiple measurements over a large number of 30s-long DASH sessions from thousands of geographically distributed IP addresses. From their measurements, we infer the download rate of each IP address for every chunk of the session and thus obtain 500,000 samples of realistic download rates. After filtering this set to remove abnormal rates, we randomly associate one download rate for every viewer. For each point in time and each channel we considered as many viewers as its popularity accordingly to our twitch dataset.

Multi Bitrate Video Transcoding. When a session is selected for a delivery by ABR streaming, the raw video is transcoded into multiple live streams. We consider the creation of one stream per resolution, only in smaller resolutions than the one of the raw video. The bitrate of each live stream depends on the bitrate of the raw video, as given in Table 1. For example, let the raw video have a bitrate of 2,000kbps and resolution 720p. The transcoded streams have a bitrate of 1,400kbps for the 480p stream, of 1,000kbps for the 360p, and of 600kbps for the 224p.

Strategies. We evaluate the *top* and *threshold* strategies against two naive and extreme strategies: the *none* strategy with no ABR implementation, and an *all* strategy where all online channels are delivered by ABR. The former represents the current state of live game streaming platforms while the latter is an upper bound of an implementation of ABR streaming. For *top* and *threshold*, we set $k = 50$ and $j = 100$ by default.

		input resolution		
		480p	720p	1080p
output res.	224p	$0.5 * n$	$0.3 * n$	$0.25 * n$
	360p	$0.7 * n$	$0.5 * n$	$0.3 * n$
	480p	n	$0.7 * n$	$0.5 * n$
	720p		n	$0.7 * n$
	1080p			n

Table 1: Bitrate of the streams transcoded from a raw video stream having bitrate n kbps

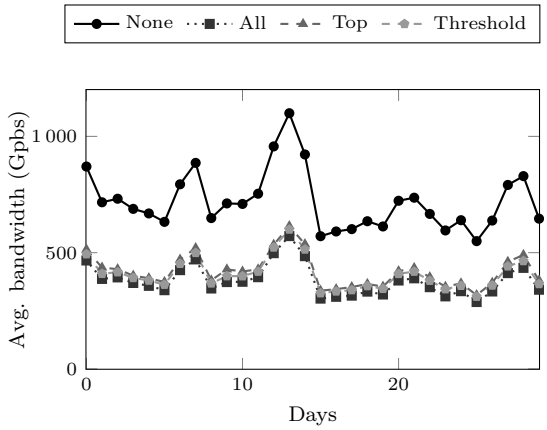
4.2 Evaluations

In Figure 5, we depict three metrics that highlight the trade-off related to the implementation of ABR on live streaming services. In Figure 5a, we measure the average daily bandwidth that is needed to deliver the video channels. The first observation is that implementing ABR generates a non-negligible reduction of the delivery. The aggregation of bandwidth savings on every degraded viewer can nearly halve the bandwidth between both extreme *none* and *all* strategies. The *top* and *threshold* strategies are close to the *all* strategy, which validates our strategy of implementing ABR streaming only for the most popular channels.

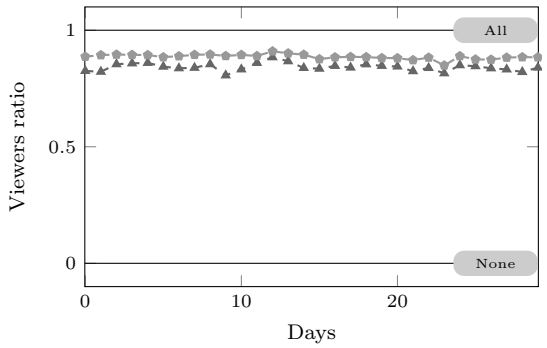
In Figure 5b, we measure the gain in QoE for the viewers. Due to lack of space, we focus on only one simple metric, which is the ratio of degraded viewers that can find in ABR streaming a video stream with a bitrate lower than their download rate. We observe in the dataset that, even with ABR, some degraded viewers have still a download rate lesser than the bitrate of the 224p stream. What we measure here is the ratio of the "addressable" degraded viewers. Again the *all* strategy gives an upper bound, and both *top* and *threshold* strategies allow most of these viewers to enjoy a video stream that fits with their network connection.

Finally in Figure 5c, we measure the computing needs with a metric in *transcoding hours* inspired from the current practice in commercial cloud transcoding offers (*e.g.* Zencoder). Both input and output resolutions are used to determine the amount of transcoding hours. For example, a given stream with 1 hour length and resolution 480p transcoded to 360p and 224p is counted as 3 hours of transcoding (1 input + 2 output). High-definition resolutions (720p and 1080p) double the transcoding hours. In Figure 5c, the main observation is that we have to use a log scale on the y-axis to keep it readable. It means that the transcoding hours of the *all* strategy is one order of magnitude larger than for the *top* and *threshold* strategies.

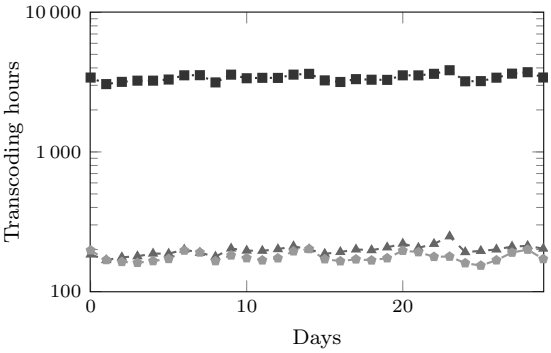
To summarize our results in a more practical way, we estimate the overall infrastructure cost for the four strategies. We use the publicly available prizes of Zencoder and Amazon FrontCloud to estimate the cost of the transcoding hours and the delivered bandwidth, respectively. We present in Figure 6 the synthesis. We see on the *all* strategy that the gains in delivery are unfortunately counter-balanced by the costs in transcoding. Both *top* and *threshold* strategies find a better trade-offs with a significant reduction of delivery cost at a negligible transcoding cost. Finally, we show that the performance gap between *top* and *threshold* strategies does not necessarily justify the implementation of *on-the-fly* strategies. Sophisticated *at-startup* strategies are expected to even reduce the gap.



(a) Bandwidth needed for streams delivery



(b) Ratio of addressed degraded viewers



(c) Transcoding hours

Figure 5: Evaluation of different metrics when implementing ABR streaming on Twitch

4.3 Playing with Strategy Parameters

We evaluate the impact of parameters k and j for the *top* and *threshold* strategies respectively. In Figure 7 we represent the total estimation costs for different values of k and j. What is interesting here is that these parameters allow Twitch to adjust the trade-off between delivery and transcoding according to any external constraint (*e.g.* price variation, data-center load and maintenance operation).

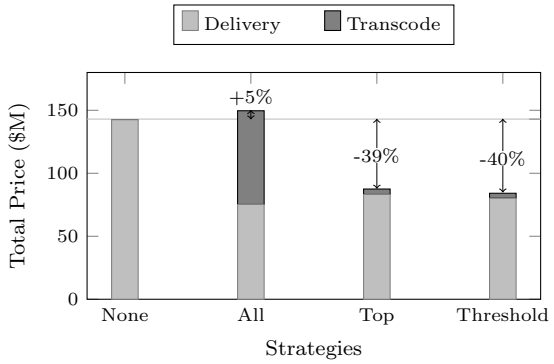


Figure 6: Estimation of the total infrastructure costs

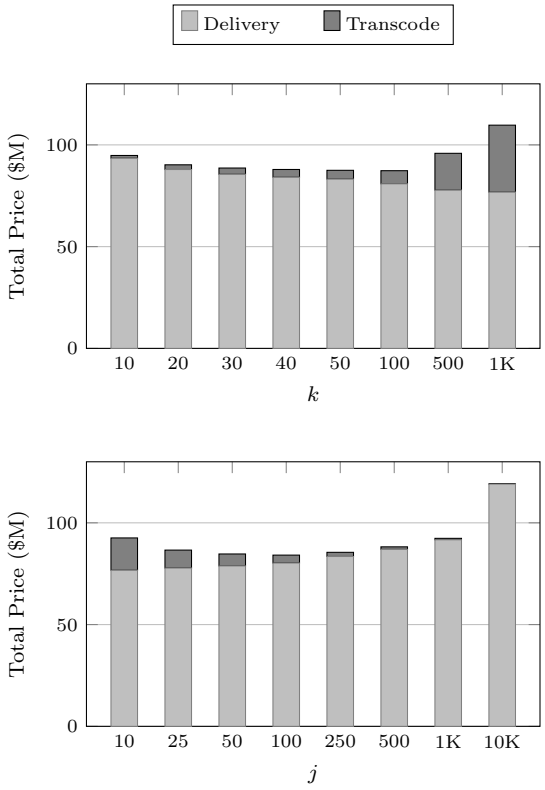


Figure 7: Estimation of the total infrastructure costs for different values of k and j in *top* and *threshold* strategies

5. DISCUSSIONS

Related Works. To the best of our knowledge, no previous work can be directly related to ours. Most studies related to live streaming platforms have dealt with Peer-to-Peer (P2P) delivery (or peer-assisted), for which both the transcoding and the delivery is under the responsibility of every broadcaster. Transcoding live stream has also been studied (including works on rate-adaptive technologies [4]) but the object of these works is not related to the management of many concurrent transcoding process. The third line of research having connection to our paper is the management of data-centers for massive UGC platforms. However no previous work has considered adaptivity in the treatment of requests, so the trade-off between computing needs and delivery cost has never been studied. Finally, Twitch has been studied in [8], but the focus is on the analysis of the behavior of a small number of professional broadcasters although we study it at a macroscopic level with a quantitative approach.

Future Works. This paper is a preliminary work in the more general context of implementing interactive multimedia services at a massive scale. We introduce an infrastructure management problem and we reveal that some simple strategies can significantly cut the overall infrastructure costs while increasing the QoE of end-users. Many future works can derive from this paper. First the optimization problem requires a formal definition and analysis. We voluntarily neglect many details in our formulation in order to provide the global picture. However a more formal and accurate formulation should include a more precise estimation of the QoE gain for the degraded viewers, a better model for the transcoding computing needs, and the management of different hardware computing resources. Second, more efficient strategies can be designed. We present here simple strategies but more sophisticated strategies are expected to yield better performances, especially for at-startup strategies. A more comprehensive analysis of the levers that make a session become popular as well as recent statistical approaches to deal with popularity forecasting represent appealing research. Third, the integration in practical platforms and real implementation also bring additional difficulties, from a technical perspective with, for example, the integration of standard, but also from a business perspective. Typically, Twitch has recently started offering ABR for some "partner" premium broadcasters.

6. REFERENCES

[1] Guidelines for Implementation: Live Services based on DASH-IF IOPs. DASH Industry Forum, Aug. 2014. `http://is.gd/pmJZgC`.

[2] Twitch.tv: 2013 retrospective, Jan. 2014. `http://twitch.tv/year/2013`.

[3] S. Basso, A. Servetti, E. Masala, and J. C. D. Martin. Measuring DASH streaming performance from the end users perspective using neubot. In *ACM MMSys*, 2014.

[4] N. Bouzakaria, C. Concolato, and J. L. Feuvre. Overhead and performance of low latency live streaming using MPEG-DASH. In *IEEE IISA*, 2014.

[5] D. Cryan. eSports video: A cross platform growth story. Technical report, IHS Tech., June 2014. `http://is.gd/NHVdfi`.

[6] D. Fitzgerald and D. Wakabayashi. Apple Quietly Builds New Networks. Wall Street Journal, Feb. 2014. `http://is.gd/MXc2b7`.

[7] F. Guillemin, B. Kauffmann, S. Moteau, and A. Simonian. Experimental analysis of caching efficiency for youtube traffic in an isp network. In *IEEE ITC*, 2013.

[8] M. Kaytoue, A. Silva, L. Cerf, W. M. Jr., and C. Raïssi. Watch me playing, i am a professional: a first study on video game live streaming. In *ACM WWW Conf.*, 2012.

[9] N. Weil. The State of MPEG-DASH Deployment. Technical report, Streaming Media, Apr. 2014. `http://is.gd/8Pcu78`.